

# Drawing Feynman Diagrams with L<sup>A</sup>T<sub>E</sub>X and METAFONT

Thorsten Ohl\*

Technische Hochschule Darmstadt  
Schloßgartenstr. 9  
D-64289 Darmstadt  
Germany

May 1995

## Abstract

**feynMF** is a L<sup>A</sup>T<sub>E</sub>X package for easy drawing of professional quality Feynman diagrams with METAFONT (or METAPOST). **feynMF** lays out most diagrams satisfactorily from the structure of the graph without any need for manual intervention. Nevertheless all the power of METAFONT (or METAPOST) is available for the most complicated cases.

---

\*e-mail: Thorsten.Ohl@Physik.TH-Darmstadt.de

## 1 Introduction

In recent years, **T<sub>E</sub>X** [?] and **L<sup>A</sup>T<sub>E</sub>X** [?] (or other macro packages for structured markup on top of **T<sub>E</sub>X**) have revolutionized the way we share information in theoretical physics (and other areas). Not only does **T<sub>E</sub>X** provide typographical capabilities which transcend those of commercial “word processors” substantially, **T<sub>E</sub>X** documents are also completely portable among computer systems. Since implementations are available for essentially all computers in use in the physics community, documents can be shared without the usual restrictions of proprietary data formats. This has enabled us to collaborate on papers with colleagues on the other side of the globe, to replace the mailing of hard copy preprints by electronic transmission and to submit these papers electronically to the publisher.

**T<sub>E</sub>X**’s portability comes with a price, though. It does deliberately not address the issue of graphical information, apart from the rudimentary (but very useful) capabilities of the **L<sup>A</sup>T<sub>E</sub>X** **picture** environment and similar packages [?]. More complex graphics can only be handled by inclusion of more or less device dependent external graphics files.

More recently, the inclusion of graphics files in the **PostScript** [?] page description language has emerged as a de facto standard. This approach restricts the portability of documents to installations where **PostScript** printers (or emulators) are available. The popularity of such devices makes this an almost moot point, though.

Nevertheless, handling graphics in an environment completely different from the (**T<sub>E</sub>X**) text environment causes other problems. Some popular packages that employ graphical user interfaces will force **PostScript** fonts for labeling on the user. These fonts will usually not blend smoothly with other fonts used in text and equations. More importantly, these packages usually lack the ability to create complex mathematical expressions which would be useful in the labels of figures in physics papers. Finally, these tools are usually less than portable to the extent that changing jobs means changing tools.

Currently there are a couple of tools available that address one or more of the above points in the context of drawing Feynman diagrams, which form one of the most frequent classes of graphics in physics papers. Michael Levine’s **feynman** package [?] is implemented on top of the standard **L<sup>A</sup>T<sub>E</sub>X** **picture** environment [?]. This makes it completely portable, but the graphics output is less than perfect. This is not the fault of the **feynman** package, but rooted in principle limitations of the **picture** environment. Jos Vermaseren’s **axodraw** package [?] uses **\special** to access **PostScript** primitives for drawing diagrams. This approach is inherently not portable (the mentioned ubiquity of **PostScript** printers makes this a minor point, though) but very flexible and produces substantially more pleasant graphics. Both packages take no advantage of the formal structure of Feynman graphs, but require the user to specify the layout manually using low level graphics primitives.

It is possible to go one step further and move from low-level tools working on points and curves to a high-level markup system working on the mathematical structure of graphs. This step will free the user from having to think about the layout and allow him to concentrate on the structure of the graph instead.

In this paper I will describe such a system, **feynMF**, which is *completely* portable among **T<sub>E</sub>X** installations. It is unique among packages for drawing

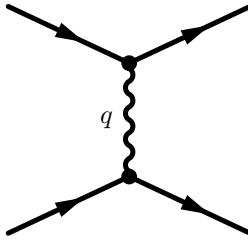


Figure 1: Simple scattering diagram.

Feynman diagrams in *combining* the following features:

- Simplicity and conciseness for common diagrams. The scattering diagram in figure 1 can be specified *completely* in five lines of L<sup>A</sup>T<sub>E</sub>X:

```
\begin{fmfchar*}(40,30) \fmfpen{thick}
  \fmfleft{i1,i2} \fmfright{o1,o2}
  \fmf{fermion}{i1,v1,o1} \fmf{fermion}{i2,v2,o2}
  \fmf{photon,label=$q$}{v1,v2} \fmfdot{v1,v2}
\end{fmfchar*}
```

It is never necessary to draft the diagram on graph paper or to perform calculations to determine the position of vertices manually.

- Expressiveness for arbitrarily complex diagrams (see the examples below).
- Extensibility.
- Portability. No graphics devices are needed beyond a standard T<sub>E</sub>X installation.
- Arbitrary T<sub>E</sub>X-labels.

The paper is structured as follows: I begin by describing the design of **feynMF** in section 2. Then I describe some details of the implementation in section 3. After a brief discussion of the most important user commands in section 4, I conclude.

## 2 Design

A clear cut distinction between “design” and “implementation” is certainly fictitious. As in most programs with more than several hundred lines of code, designs have been adapted as implementation progressed and feedback from early users came in.

### 2.1 Goals

As mentioned in the introduction, **feynMF** was to meet the following competing design goals:

- convenience and ease-of-use,

- expressiveness,
- extensibility, and
- portability.

Of these, extensibility is not a goal in itself but should rather be viewed as a derived goal. It appears impossible to reconcile ease-of-use with expressiveness in the straight jacket of an inextensible implementation. It is much more effective to provide an extensible environment where simple building blocks can be used for the straightforward solution of simple problems *and* can be combined to solve the most complicated problems, once the software system has been mastered by the user.

The reconciliation of convenience and expressiveness was made possible by providing two different modes:

- *graph mode*, in which the layout is determined automatically from a simple mathematical description of the graph, and
- *immediate mode*, in which the user has complete freedom but at a basic familiarity with METAFONT is recommended.

The goal of portability was easily reached by basing the implementation of TEX and METAFONT, because both programs will be available to all potential users of the software.

## 2.2 Languages

The primary user interface is a set of LATEX macros. It is therefore possible to keep the whole paper, including graphics, in a single file. This is, among other things, very convenient for exchanging manuscripts by electronic mail [?]. Also, no new syntax has to be learned by the user.

METAFONT [?] (or alternatively METAPOST [?]) has been chosen as the low level graphics engine for the following reasons:

- METAFONT is part of any reasonable TEX installation, therefore available to all potential users,
- METAFONT output is readily included in TEX documents in the form of (unusual) characters,
- METAFONT has very powerful graphics primitives [?], which allow high quality output, and
- METAFONT has builtin linear algebra [?], which can be employed for automatic layout algorithms, as detailed below.

Without taking advantage of these features, the implementation in other languages would have been much more complex.

### 2.3 Algorithmic layout

Early in the design it was clear that **feynMF** should in at least one mode of operation accept a *mathematical* description of a graph and create the layout of the corresponding Feynman diagram automatically. It should also not rely on a database of common topologies, because such a database will necessarily remain incomplete.

Every graph can be specified completely by giving a set  $A$  of pairs. The set  $V$  of vertices is then given by

$$V = \{ v \mid \exists a \in A : a = (v, v') \vee a = (v', v) \} . \quad (1)$$

The set  $A$  will henceforth be called the set of arcs. It is useful to introduce the sets of vertices connected to  $v$

$$\alpha(v) = \{ v' \in V \mid v * v' \} , \quad (2)$$

where  $*$  denotes the symmetrical relation

$$(v * v') \Leftrightarrow (\exists a \in A : a = (v, v') \vee a = (v', v)) \quad (3)$$

of being connected.

The obvious first candidate for a function that should be minimized is the sum of the squared lengths of arcs

$$l(v_1, \dots, v_n) = \sum_{\substack{i,j=1 \\ v_i * v_j}}^n (v_i - v_j)^2. \quad (4)$$

As is stands, (4) is not yet sufficient, because  $v_i = \hat{v}$  for all  $i$  is obviously a solution corresponding to the minimum  $l = 0$  for all  $\hat{v}$ . In order to lift the degeneracy, we have to break translational invariance. However, breaking the translational invariance by demanding for instance that the center of gravity  $\sum_{i=1}^n v_i / n$  coincides with the center of the picture is still not enough, because we hardly want all arcs to shrink to a point.

It will now be useful to introduce the set of all external vertices

$$V^{\text{ext}} = \{ v \in V \mid |\alpha(v)| = 1 \} \quad (5)$$

and its complement, the set of all internal vertices

$$V^{\text{int}} = V \setminus V^{\text{ext}}. \quad (6)$$

From (4) we see that the vertices in  $V^{\text{ext}}$  will occupy the same position as their single neighbor, unless their position is fixed explicitly. It is therefore necessary to specify explicit positions for the external vertices. In the implementation of **feynMF**, commands are provided to place a list of external vertices on “galleries” along the sides of the diagram. Using a similar strategy for external vertices, (4) has been used for example in [?] with good results for automated drawing of tree diagrams in PostScript.

While (4) gives fair results for almost all tree diagrams, it can fail miserably on loop diagrams, as witnessed in figure 2a. A simple generalization of (4) can

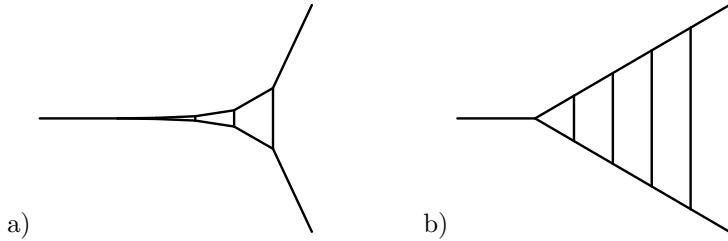


Figure 2: Ladder diagram, a) using the “action” (4) and b) using the improved “action” (7).

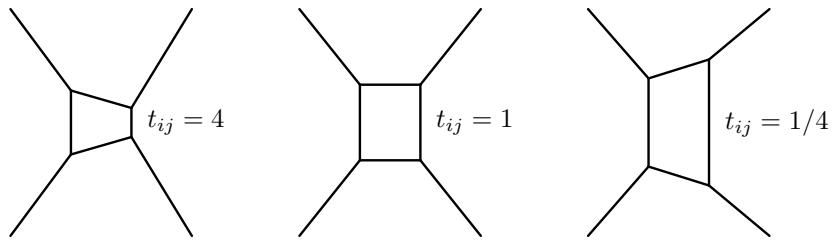


Figure 3: Varying the tension parameter.

improve results immensely

$$L(v_1, \dots, v_n) = \frac{1}{2} \sum_{\substack{i,j=1 \\ v_i * v_j}}^n t_{ij} (v_i - v_j)^2. \quad (7)$$

The elements of the symmetrical “tension” matrix  $t_{ij}$  are positive numbers that default to 1 and can be used to tune the layout.

The effect of the tension parameter can be understood by imagining the graph as consisting of rubber bands. Changing the tension of an arc will pull adjacent vertices together or allow them to move apart. As an example, figure 3 shows the effect of varying the tension of one line from 4 to 1/4.

The improved ladder diagram in figure 2b has been drawn with vanishing tension of the arcs, which will result in straight lines for the stems.

In fact, the effect of vanishing tension can also be achieved by laying out subgraphs step by step. By freezing the layout of the subgraph excluding the rungs in figure 2b first and adding the rungs later, we arrive at the same result. Obviously this procedure can be iterated for graphs of arbitrary complexity.

While there are also commands to fix the position of a vertex or to shift its position, it turns out that the most effective way of drawing Feynman diagrams consists in a combination of the stepwise construction of subgraphs and adjustment of tensions. User’s discretion is advised in tuning tension parameters. More often than not, the defaults give satisfactory results that can be made perfect by adjusting the tension of a single arc or loop. Tuning too many tensions is not likely to improve the results and is almost as time consuming as choosing the layout manually.

Technically, the most convenient aspect of (7) is that minimizing it leads to *linear* equations (see (11) below), which are easily solved. It would in principle be possible to investigate improved functions like

$$N(v_1, \dots, v_n) = \sum_{\substack{i,j=1 \\ v_i \neq v_j}}^n ((v_i - v_j)^2 - \delta^2)^2, \quad (8)$$

which would avoid the problem of figure 2a to some extent by favoring arcs of length  $\delta$ . However, the prize in having to solve a system of non-linear equations is certainly too high, in particular because it will be impossible to prove that reasonable results will result from *all* user input.

## 2.4 Constraints

The method of Lagrange multipliers allows us to specify linear constraints among vertices

$$v_i - v_j = d_{ij}, \quad (9)$$

while still dealing with linear equations. Therefore we add the term

$$\Lambda = \sum_{\substack{i,j=1 \\ v_i \sim v_j}}^n \lambda_{ij} \cdot (v_i - v_j - d_{ij}) \quad (10)$$

to the “action” (7). Then the system of  $2n_\alpha + 2n_\gamma$  linear equations determining the layout is

$$0 = \frac{\partial}{\partial v_i^\mu} (L + \Lambda) = \sum_{\substack{j=1 \\ v_i \neq v_j}}^n t_{ij} (v_i^\mu - v_j^\mu) + \sum_{\substack{j=i+1 \\ v_i \sim v_j}}^n \lambda_{ij}^\mu - \sum_{\substack{j=1 \\ v_i \sim v_j}}^{i-1} \lambda_{ji}^\mu \quad (11)$$

$$0 = v_i^\mu - v_j^\mu - d_{ij}^\mu \quad | \quad v_i \sim v_j.$$

Experience shows that non-linear constraints like fixing a distance

$$|v_i - v_j| = \delta \quad (12)$$

would be useful sometimes, but as discussed at the end of section 2.3, their implementation is beyond the scope of **feynMF**.

## 2.5 Immediate mode

In addition to the graph mode for algorithmic layout that has been described in the previous section, **feynMF** also has an immediate mode to provide the user with maximum flexibility. Immediate mode is particularly useful for unusually curved arcs, which can not be specified easily in graph mode. In this mode, arbitrary METAFONT paths can be drawn, either specified by absolute coordinates or derived from arcs entered previously in graph mode. For a detailed description of METAFONT’s features, the reader is referred to [?].

## 2.6 Extension mechanism

A great variety of different line styles is in use in the physics community. `feynMF` provides the most common of them per default, as displayed in table 1. While it would at best be inefficient to support an exhaustive list of such styles, it would probably be a futile effort anyway. Instead, `feynMF` implements an extension mechanism that allows the user to install custom line styles of arbitrary complexity. For this purpose, a macro `style_def` is provided. This macro defines a METAFONT function that does the actual drawing based on the path it receives as an argument. Furthermore it records the name of the function to make it available to graph mode and immediate mode as well.

# 3 Implementation

`feynMF` is implemented in the form of two macro packages: `feynmf.sty` for the L<sup>A</sup>T<sub>E</sub>X part and `feynmf.mf` for the METAFONT part. Let us consider both of them in turn.

## 3.1 L<sup>A</sup>T<sub>E</sub>X macros

Most macros in `feynmf.sty` are trivial in that they are just writing their METAFONT equivalent to the METAFONT input file. The `\fmf` macro, for example, is just the T<sub>E</sub>X version of the `vconnect` METAFONT function:

```
\def\fmf#1#2{\fmfcmd{vconnect ("#1", \pfx{#2});}}
```

Here `\fmfcmd` writes its expanded argument to the METAFONT file and `\pfx` adds a “`_`” prefix to each member of the comma separated list it takes as an argument. This measure protects the unwary user from the mysterious errors caused by accidentally using a METAFONT reserved word for a vertex name.

A non-trivial aspect of the `\pfx` macro worth mentioning is that it works by macro expansion alone (in T<sub>E</sub>X’s “mouth” in Knuth’s terminology [?]) and does not need to redefine any macro (an operation that would have to happen in T<sub>E</sub>X’s “stomach” in Knuth’s terminology). This is necessary for making `\pfx` work inside of a `\write`, where macros are expanded but redefinitions are prohibited (see [?], Appendix D). Traditional implementations of looping constructs (e.g. `\loop ... \repeat`) work by redefining a continuation and are therefore unavailable inside of a `\write`. A possible solution would be to use a temporary variable and force expansion of `\pfx` outside of the `\write`. A far more elegant solution uses a subset of a partial implementation [?] of  $\lambda$ -calculus [?] in T<sub>E</sub>X’s “mouth”.

The other unusual aspect of the L<sup>A</sup>T<sub>E</sub>X macros is the

```
\grepfile{\langle pattern\rangle}{\langle infile\rangle}{\langle outfile\rangle}
```

macro that copies all lines starting with `:⟨pattern⟩:` from `⟨infile⟩` to `⟨outfile⟩` after stripping off the `:⟨pattern⟩:`. It is used to extract the label information that the METAFONT macros have stored in the `log`-file. This trick overcomes METAFONT’s limitation of not being able to open any other files than the terminal, the `gf`-file, the `tfm`-file, and the `log`-file. The implementation of this macro is straightforward using T<sub>E</sub>X’s pattern matching macro definitions.

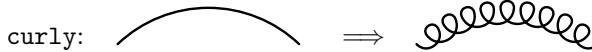
However, all subtleties of these  $\text{\LaTeX}$  macros are of no concern to the user, because they are designed to do their work quietly “behind the scenes”.

### 3.2 METAFONT macros

The METAFONT macros are much richer than their  $\text{\LaTeX}$  counterparts. They have to deal with drawing primitives, linear algebra and abstract representations of graphs.

#### 3.2.1 Transformers

An important tool for generating complex graphs with arcs of different styles is provided by transformers. These are functions that take a simple path (determined from the layout algorithm or specified explicitly) as argument and return another path which corresponds to a decorated version. Here is an example that is used for implementing gluon lines:



Using similar transformers, the implementation of dedicated drawing functions is a matter of combining simple building blocks:

```
style_def gluon_with_arrow expr p =
  draw (wiggly p);
  fill (arrow p)
enddef;
```



The current implementation does not attempt to force decorations (e.g. arrows) into the transformer paradigm, because METAFONT treats drawing along a path differently from filling an outline. Therefore decorations are drawn after each other and not added to the object in a pipeline.

#### 3.2.2 Graphs

Graphs are represented by an array of vertices and arrays of vertices emanating from the vertices. Therefore the core of the data structure for graphs is given by

```
numeric vlist.first;
numeric vlist.last;
pair vlist[]loc;
numeric vlist[]arc.first;
numeric vlist[]arc.last;
numeric vlist[]arc[];
numeric vlist[]arc[]tns;
```

Here  $vlist[i].loc$ , with  $vlist.first \leq i \leq vlist.last$ , is an array of two dimensional coordinates, one for each vertex. These coordinates start in the state `unknown` and become `known` when the layout equations have been solved. For each vertex  $i$ ,  $vlist[i].arc[j]$ , with  $vlist[i].arc.first \leq j \leq vlist[i].arc.last$ , is an array of numbers pointing to another vertex. Therefore, each entry corresponds to an arc. The  $vlist[i].arc[j].tns$  array holds the elements  $t_{ij}$  of the tension matrix.

This data structure is sufficient for performing the algorithmic layout, as described below. It is supplemented by similar arrays holding information on linear constraints, line styles, etc.

### 3.2.3 Linear algebra

Let us now discuss how to solve the layout equation (11) for the common case of no constraints

$$0 = \sum_{\substack{i,j=1 \\ v_i \neq v_j}}^n t_{ij} (v_i^\mu - v_j^\mu). \quad (11')$$

Adding the constraints is a straightforward exercise, which is omitted here for brevity. METAFONT's syntactical features and builtin linear algebra allow a direct translation of (11'):

```
for i = vlist.first upto vlist.last:
  if unknown vlist[i].loc:
    origin = origin
    for j = vlist[i].arc.first upto vlist[i].arc.last:
      + vlist[i].arc[j].tns * (vlist[i].loc - vlist[vlist[i].arc[j]].loc)
    endfor;
  fi
endfor
```

METAFONT's syntactical feature that allows this translation of (11') is the decoupling of control structures (`for ... endfor`, `if ... fi`) from mathematical expressions. This means that the bodies of loops and conditionals do not have to form syntactically complete expressions. We can therefore use loops to construct expressions from building blocks. For the example of figure 1 the above fragment expands to

```
origin = origin
+ vlist[5].arc[1].tns * (vlist[5].loc - vlist[1].loc)
+ vlist[5].arc[2].tns * (vlist[5].loc - vlist[2].loc)
+ vlist[5].arc[6].tns * (vlist[5].loc - vlist[6].loc);
origin = origin
+ vlist[6].arc[3].tns * (vlist[6].loc - vlist[3].loc)
+ vlist[6].arc[4].tns * (vlist[6].loc - vlist[4].loc)
+ vlist[6].arc[5].tns * (vlist[6].loc - vlist[5].loc);
```

since the vertices  $v_1, v_2, v_3, v_4$  are already fixed as external vertices. If all tensions are unity, this is the linear system

$$\begin{aligned} 0 &= 3v_5 - v_1 - v_2 - v_6 \\ 0 &= 3v_6 - v_3 - v_4 - v_5 \end{aligned} \quad (13)$$

with unique solution

$$\begin{aligned} v_5 &= \frac{1}{8} (3v_1 + 3v_2 + v_3 + v_4) \\ v_6 &= \frac{1}{8} (3v_3 + 3v_4 + v_1 + v_2) . \end{aligned} \tag{13'}$$

Note that we do not have to find the solution ourselves, because **METAFONT** will not interpret the equations as assignments, but rather as linear equations. Once enough equations are given, the state of the vertex coordinate will change from **unknown** to **known** and will have a value. As long as all vertices belong to a subgraph with at least one element in  $V^{\text{ext}}$ , there will be a unique solution to the layout equations (11).

### 3.2.4 Labels

An interesting feature of **feynMF** is the ability to calculate optimal label positions in **METAFONT** and to communicate this information back to **LATEX**'s **picture** information. Because **METAFONT** can not write any other files than its **log**-file, the information has to be stored there and **TeX** macros have to be used to parse this file.

The algorithm used is quite simple. It will place all labels on the outside of the arc or vertex it is associated to. If the result is not satisfactory, explicit placement rules can be specified to overwrite the automatic layout.

### 3.2.5 Immediate mode

The implementation of immediate mode is fairly straightforward, because all necessary drawing primitives for drawing **feynMF**'s line styles on **METAFONT** paths are already available for graph mode. Therefore only trivial **LATEX** macros have to be written that translate the **LATEX** syntax to **METAFONT**.

### 3.2.6 Extension mechanism

The extension mechanism serves two major purposes: it allows users to specify new line styles and to overload existing line styles.

The ability to overload line styles can be used for a purely symbolic markup of graphs. If all the arcs are tagged by symbolic names like **gluon**, **technipion**, **chargino**, etc., each user can use a library of style definitions to render the graph in a customized visual appearance.

## 4 Usage

Here is a short summary of the most important user commands of **feynMF**. This section is not intended to replace the user's manual that comes with the distribution [?], but it should give nevertheless an idea how **feynMF** is used.

The overall structure is controlled by two environments:

```
\begin{fmffile}{\name} ... \end{fmffile}
```

This environment encloses all graphs that are written into a **METAFONT** input file named `\name.mf`. For technical reasons, `\name`

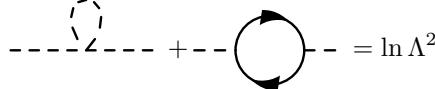
*must not* be identical to the name of the main L<sup>A</sup>T<sub>E</sub>X input file. All created files have to be processed by METAFONT after the first run of L<sup>A</sup>T<sub>E</sub>X. See the **feynMF** user's manual for details on how to run METAFONT on various systems.

```
\begin{fmfgraph*}(<w>,<h>) ... \end{fmfgraph*}
```

This environment encloses a single graph of width  $\langle w \rangle$  and height  $\langle h \rangle$  L<sup>A</sup>T<sub>E</sub>X \unitlengths. There is also a light weight unstarred version that does not support labels.

These environments can be used everywhere in a L<sup>A</sup>T<sub>E</sub>X document, in particular in centered \parboxes for creating graphical equations like

```
\parbox{20mm}{\begin{fmfgraph}(20,15)
  \fmfleft{i} \fmfright{o} \fmf{dashes}{i,v,v,o}
\end{fmfgraph}} +
\parbox{20mm}{\begin{fmfgraph}(20,15)
  \fmfleft{i} \fmfright{o} \fmf{dashes}{i,v1} \fmf{dashes}{v2,o}
  \fmf{fermion,left,tension=.3}{v1,v2,v1}
\end{fmfgraph}} = \ln\Lambda^2
```



## 4.1 Graph mode

The placement of external vertices is controlled by the following commands:

```
\fmfleft{<v1>[, ... ]}
```

place the vertices  $\langle v_1 \rangle, \dots$  equidistantly on a smooth path on the left side of the diagram. There are analogous **\fmfright**, **\fmftop**, **\fmfbottom**, and **\fmfsurround** commands. The latter will place the vertices on a smooth path surrounding the diagram.

```
\fmfleftn{<v>}{<n>}
```

place the vertices  $\langle v_1 \rangle, \dots, \langle v_n \rangle$  equidistantly on a smooth path on the left side of the diagram. Analogously for **\fmfrightn**, **\fmftopn**, **\fmfbottomn**, and **\fmfsurroundn**.

The external vertices can be connected with themselves and internal vertices by the commands

```
\fmf{<style>[,<opt>[, ... ]]}{<v1>,<v2>[, ... ]}
```

connect the vertices  $v_1, v_2, \dots$  by a line of style **<style>** with options **<opt>** switched on. The available line styles are collected in table 1. Additional styles can be defined with **style\_def**. A special line style is **phantom**, which will not draw an arc at all. It is nevertheless useful for manipulating the layout, because the corresponding arc enters the layout equations. For convenience and for allowing more descriptive specifications, several aliases like **gluon**, **quark**, or **photon** of the line styles in table 1 are defined. Among the options, which can be given in a comma separated list after the line style are

Name	Example	Parameters
curly		curly_len
dbl_curly		curly_len
dashes		dash_len
dashes_arrow		dash_len
dbl_dashes		dash_len
dbl_dashes_arrow		dash_len
dots		dot_len
dots_arrow		dot_len
dbl_dots		dot_len
dbl_dots_arrow		dot_len
phantom		
phantom_arrow		
plain		
plain_arrow		
dbl_plain		
dbl_plain_arrow		
wiggly		wiggly_len
dbl_wiggly		wiggly_len
zigzag		zigzag_width
dbl_zigzag		zigzag_len

Table 1: Available line styles

**tension:** change the tension matrix element  $t_{ij}$  from the default value of 1.

**left, right:** draw on a half circle on the left or right.

**label:** arbitrary TeX commands for labeling the arc (macros should be protected with `\noexpand`).

**label.side, label.dist:** force placement of the label. on the left or right at this distance

**width:** change the width of the line.

**foreground, background:** colors (available with METAPOST only!).

`\fmfn{<style>[,<opt>[, ... ]]}{<v>}{<n>}`  
 connect the vertices  $v_1, \dots, v_n$  by a line of style  $<style>$  with options  $<opt>$  switched on.

The decoration of vertices is affected by the commands:

`\fmfv{ $\langle opt \rangle$ }[, ... ]}{ $\langle v_1 \rangle$ [, ... ]}`  
 turn on the options  $\langle opt \rangle$  for the vertices  $v_1, \dots$ . Among them are

`label`: arbitrary TeX commands for labeling the vertex (macros should be protected with `\noexpand`).  
`label.angle, label.dist`: force placement of the label at this angle or distance.  
`decoration.size, decoration.filled`: size and filling style of the decoration.  
`decoration.shape, decoration.angle`: shape of the decoration, optionally rotated.  
`foreground, background`: colors (available with METAPOST only!).

Here are some examples for vertex decorations:

shaded (`fill=.5`) circle :  and hatched (`-.5`) square: 

open (`fill=0`) triangle, diamond, pentagon, and hexagon:



filled (`fill=1`) triagram, tetragram, pentagram, and hexagram:



`\fmfvn{ $\langle opt \rangle$ }[, ... ]}{ $\langle v \rangle$ }{ $\langle n \rangle$ }`  
 turn on the options  $\langle opt \rangle$  for the vertices  $v_1, \dots, v_n$ .

`\fmfblob{ $\langle d \rangle$ }{ $\langle v_1 \rangle$ [, ... ]}`  
 draw a blob of diameter  $\langle d \rangle$  at the vertices  $v_1, \dots$ . There is an analogous `\fmfblobn` command.

`\fmfdot{ $\langle v_1 \rangle$ [, ... ]}`  
 draw a dot at the vertices  $v_1, \dots$ . There is an analogous `\fmfdotn` command.

`\fmflabel{ $\langle label \rangle$ }{ $\langle v_1 \rangle$ [, ... ]}`  
 label the vertices  $v_1, \dots$  with  $\langle label \rangle$ .

The location of vertices can be fixed, but experience shows that this command should only be used as a last resort:

`\fmfforce{ $\langle z \rangle$ }{ $\langle v_1 \rangle$ [, ... ]}`  
 place the vertices  $v_1, \dots$  at the METAFONT coordinate  $\langle z \rangle$ .

The layout calculation and drawing, which are implicitly performed at the end of each `fmfgraph`, can be forced by

`\fmffreeze`  
 freeze the positions of the vertices entered so far.

```
\fmfdraw
  draw all arcs and vertices entered so far.
```

The appearance of the graph can be changed by

```
\fmfpen{w}
  change the width of the drawing “pen” to w. The default is thin = 1 pt.
```

```
\fmfset{par}{val}
  set the parameter par to the value val.
```

```
\fmffixed{d}{v1}[,...]
  fix the distance vector between subsequent vertices in the list v1, ... to d.
```

## 4.2 Immediate mode

In immediate mode, the drawing commands from graph mode are duplicated with different arguments. Therefore all decorated line styles are available, but now for arbitrary METAFONT paths.

```
\fmfi{style}[,{opt}][,...]{path}
  draw a line of style style on the METAFONT path path with options opt switched on.
```

```
\fmfiv{opt}[,,...]{z}
  draw a vertex with options opt at the METAFONT coordinate z.
```

```
\fmfipair{var}, \fmfipath{var}
  declare the variable var as a pair (coordinate) or path.
```

```
\fmfiset{x}{y}
  assign the value of y to the variable x.
```

```
\fmfiequ{x}{y}
  declare equality of the variables x and y. This is different from assignment, because METAFONT can solve linear equations [?].
```

Immediate and graph mode can be interfaced by using the following METAFONT functions to access coordinates from graph mode in immediate mode:

```
vpath[tag](_from,_to)
  return the METAFONT path of the arc from vertex from to vertex to. The optional tag can be used to disambiguate arcs connecting the same vertices.
```

```
vloc(_v)
  return the position of the vertex in METAFONT coordinates.
```

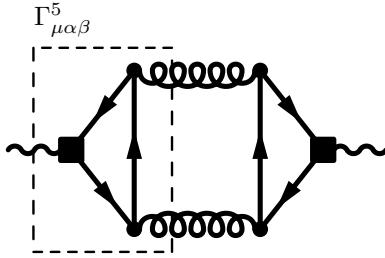


Figure 4: Three loop QCD correction to the axial vector part of the  $Z^0$ -selfenergy.

### 4.3 Extension mechanism

A powerful, but dangerous command is

```
\fmfcmd{\langle METAFONT-expression\rangle}
        write an arbitrary \langle METAFONT-expression\rangle to the METAFONT file. No
        semicolon is appended.
```

A recommended application of \fmfcmd is in defining new line styles with

```
style_def <name> expr p = ... enddef;
        define a new line style and register it as <name>.
```

here is an example that will draw a cross at the center of the arc:

```
\fmfcmd{%
vardef cross_bar (expr p, len, ang) =
((-len/2,0)--(len/2,0))
    rotated (ang + angle direction length(p)/2 of p)
    shifted point length(p)/2 of p
enddef;
style_def crossed expr p =
    cdraw p;
    ccutdown cross_bar (p, 5mm, 45);
    ccutdown cross_bar (p, 5mm, -45)
enddef;}
```

as in

```
\fmfcmd{crossed}{v1,v2} ==> —————×————
```

### 4.4 Advanced example

Finally, figure 4 is a slightly more advanced example: a three loop QCD correction to the axial vector part of the  $Z^0$ -selfenergy. The beginning is straightforward: we connect the vertices. It is beneficial to give a higher tension to the bosons and a lower tension to the gluons to balance the diagram:

```
\fmfpen{thick} \fmfleft{i} \fmfright{o}
\fmf{boson,tension=2}{i,iv3} \fmf{boson,tension=2}{o,ov3}
\fmf{quark}{iv1,iv2,iv3,iv1} \fmf{quark}{ov1,ov2,ov3,ov1}
\fmf{gluon,tension=.5}{ov1,iv1} \fmf{gluon,tension=.5}{iv2,ov2}
```

Now we add dots to the vector vertices and big squares to the axial vector vertices:

```
\fmfv{decor.shape=square,decor.size=4thick}{iv3,ov3}
\fmfdot{iv1,iv2,ov1,ov2}
```

As it stands, the diagram will have all vertices on a straight line. To remedy this situations, we use `\fmffixed` to open the triangles:

```
\fmffixed{(0,.7h)}{iv1,iv2} \fmffixed{(0,.7h)}{ov1,ov2}
```

We could also have used `phantom` arcs to achieve similar results, but here the linear constraints are more concise and intuitive.

For illustration, we mark the left triangle subgraph by a dashed box. As of version 1.0, `feynMF` does not have a builtin function for calculating the enclosing box of a list of vertices. However, this is easily done in a few lines of METAFONT. Before we start, we have to force the layout calculation:

```
\fmffreeze
```

For convenience, we store the coordinates of the vertices in temporary variables:

```
\fmfcmd{%
  save loc, bmin, bmax;
  forsuffixes $ = 1, 2, 3:
    (loc.$.x, loc.$.y) = vloc __iv.$;
  endfor
}
```

It is trivial to get the coordinates of the enclosing box:

```
bmax.x = max (loc1x, loc2x, loc3x) + .1w;
bmax.y = max (loc1y, loc2y, loc3y) + .1h;
bmin.x = min (loc1x, loc2x, loc3x) - .1w;
bmin.y = min (loc1y, loc2y, loc3y) - .1h;}
```

Now we can use immediate mode to draw this box, thinly dashed and labeled

```
\fmfi{dashes, width=thin}{%
  (bmin.x,bmin.y) -- (bmax.x,bmin.y)
  -- (bmax.x,bmax.y) -- (bmin.x,bmax.y) -- cycle}
\fmfiv{label=$\Gamma_5_{\mu\alpha\beta}$}%
{(.5[bmin.x,bmax.x],bmax.y)}
```

## 5 Conclusions

I have described `feynMF`, a flexible tool for portable and convenient inclusion of Feynman diagrams in L<sup>A</sup>T<sub>E</sub>X documents.

## Acknowledgments

I am most grateful to Wolfgang Kilian, who pushed `feynMF`'s predecessor `feynman.mf` to its limits in his doctoral thesis and provided invaluable input for `feynMF`. Thanks also to all brave users who tested preview versions and provided encouragement.

## A Distribution

The latest release of **feynMF** is available by anonymous ftp from

`crunch.ikp.physik.th-darmstadt.de`

in the directory

`pub/ohl/feynmf`

or from any of the Comprehensive T<sub>E</sub>X Archive Network (CTAN) hosts

`ftp.shsu.edu, ftp.tex.ac.uk, ftp.dante.de`

in the directory

`macros/latex/contrib/supported/feynmf`

Important announcements (new versions, fatal bugs, etc.) will be made through the mailing list

`feynmf-announce@crunch.ikp.physik.th-darmstadt.de`

Subscriptions can be obtained from

`majordomo@crunch.ikp.physik.th-darmstadt.de`

(send a message consisting of `help` to `majordomo` for instructions on how to subscribe, don't send such messages to the list itself).

## B Installation

**feynMF** comes in standard L<sup>A</sup>T<sub>E</sub>X doc format [?]. The installation procedure is described in the `README` file and need not be repeated here.

## C Revision History

### Version 1.0, May 1995

First official release.